Congestion Avoidance and Control

Masood Khosroshahy

June 2009

Tech. Report

Contents

1 Congestion Control Principles

The need for congestion control and dangers of neglecting to apply sound congestion control mechanisms are discussed in [1]. According to [1], TCP's congestion avoidance mechanism has been credited for being one of the reasons behind the success of the Internet. Maintaining the stability of the Internet is very much dependent on the use of end-to-end congestion control by best-effort traffic, yet, an increasing number of applications[1] choose not to use TCP as transport protocol or otherwise be TCP-friendly in terms of congestion control. The main reasons for a flow to use end-to-end congestion control are: 1) Prevention of congestion collapse of the Internet 2) Achieving fairness in terms of share of the bandwidth that is obtained by the flow among competing flows in a congested link 3) Optimizing its own performance metrics such as throughput, delay and loss.

The phenomenon of the Internet Congestion Collapse[2] was first experienced in mid 1980s which prompted the addition of congestion avoidance mechanism to TCP which ensured that TCP flows "back off" in response to congestion signals (dropped packets). Stability of the Internet, to a large extent, depends on end-to-end congestion control as well as on router-based mechanisms which are especially important in the face of a growing number of unresponsive flows (i.e. flows that do not use end-to-end congestion control or otherwise do not back off in response to congestion).

A concern that was raised in the late 1990s was the decision by Internet Browser developers to work around the congestion avoidance mechanism of TCP (which effectively slowed down transfer rate) by opening multiple TCP connections to the same server. This issue was addressed in [5] by providing guidelines for the maximum number of simultaneous persistent connections (with maximum being two) that a client can open and maintain to any given destination/server. While main Internet Browser vendors might have complied with this guideline, the issue has manifested itself in another form with the advent of P2P applications with each client maintaining dozens of simultaneous connections to its peers.

Two known scenarios that can cause congestion collapse are: 1) Unnecessary retransmission of packets by TCP connections 2) Undelivered packets (waste of bandwidth is caused by transferring the packets through the network only to be dropped before arriving at the destination). The first

[1] Multicast traffic and unicast traffic like streaming multimedia, to name a few.

[2] The "Congestion Collapse" event reported in [2] with later consequent modifications to TCP detailed in [3, 4].

scenario was largely responsible for the first congestion collapse experienced in the mid 1980s and has been resolved to a large extent by improvements in timer and other congestion control mechanisms in recent implementations of TCP. However, the second scenario has remained unresolved and threatens the stability of today's Internet. Proliferation of applications not using end-to-end congestion control, P2P applications with numerous simultaneous connections and applications that increase their sending rate in response to increased drop rate in the network are breathing new life into a scenario of incipient Internet congestion collapse. End-to-end congestion control by the end nodes, the network guaranteeing the final delivery of accepted packets at congested links and adherence of application developers to a non-aggressive use of the network could address the concerns regarding congestion collapse of today's Internet.

As detailed in [6], traditional "Drop-Tail" buffer management in router buffers has two main shortcomings: 1) A single flow, or few flows, may take up the whole buffer space which results in all other flows being locked out (which is a major fairness concern) 2) Drop-Tail policy allows full or near-full steady-state buffer occupancy which results in high end-to-end delay (which is especially detrimental to the performance of interactive applications like Telnet and web browsers). "Active Queue Management", or AQM, refers to a policy based on which a router decides to drop the arriving packets before the queue space is exhausted. Dropping packets leads to TCP sources decreasing their sending rate which eventually means decreased aggregate arrival rate to the router buffer from all the flows. AQM also leads to smaller steady-state queue sizes which presents the option for routers to accept occasional bursts of packets thanks to available space in their buffers. This in turn leads to higher link utilization as well as the previously-mentioned lower end-to-end delay. Bursts of packets arriving at a router's buffer is a frequent event and the capacity of absorbing these bursts using AQM corrects the bias against bursty traffic due to Drop-Tail policy and its resulting steady-state near-full queue size. AQM prevents also existence of a full queue which leads to simultaneous packet drops from all flows, hence global synchronization of TCP connections and periods of under-utilization of the link capacity. Finally, AQM solves the flow lock-out phenomenon, present when using Drop-Tail policy, as AQM always ensures some buffer space for incoming packets at all times.

2 TCP Congestion Control Algorithms and TCP-friendliness

TCP (Transmission Control Protocol) congestion control algorithms [4, 7] have played an important role in avoiding the Congestion Collapse [3] in the Internet. In what follows, we briefly explain how these algorithms interact to help a TCP flow avoid causing congestion in the network. The rate at which a TCP sender can transmit segments is governed by two windows; the congestion window (monitors congestion situation in the network) and the receiver's advertised window (monitors receiver side parameters such as buffer limits). Therefore, the TCP sender will be able to transmit as fast as the minimum indicated by the two windows[3].

When a TCP flow first starts, it enters a "slow-start" phase which is governed by a small initial value for the congestion window. In each roundtrip time (RTT), the TCP sender can double the congestion window and send faster, if so desired. The size of the congestion window however, is limited by the value "ssthresh" set for the "slow-start" phase. As soon as this value is reached or congestion is detected, "slow-start" phase ends. In case no congestion is detected, TCP's "Additive Increase, Multiplicative Decrease (AIMD)" algorithm takes over, entering the "congestion avoidance" phase. In this phase, in each RTT, a TCP sender can increase the congestion window by at most one segment. In case congestion is detected, the congestion window is halved. The AIMD is designed to ensure that a source "backs off" quickly in congestion situations. To detect segment loss, and congestion in the network, a TCP sender uses a "retransmit timer". If the retransmit timer expires without receiving a segment reception acknowledgement, the TCP sender infers that the segment has been lost. It then ends the AIMD phase, sets the value of "ssthresh" to half the size of the current window, goes into the "slow-start" phase and retransmits the lost segment.

When an out-of-order segment arrives at a TCP receiver, the receiver sends a duplicate ACK for the last segment that was received in order. If a TCP sender receives three such duplicate ACKs, it infers a segment loss. As some segments have already been received, although out-of-order, there is no need to re-entering the "slow-start" phase. Instead, the TCP sender enters the "fast retransmit and fast recovery" phase. It retransmits the segment that appears to have been lost without waiting for the retransmit timer. After this retransmission and until a non-duplicate ACK is received, "fast

[3]The "receiver's advertised window" rate control is usually referred to as "flow control", compared to "congestion control" which is used to describe the function of "congestion window".

recovery" algorithm takes over. In this phase, the "ssthresh" is set to half the size of the current congestion window, the window is subsequently halved and then increased by three segments size, taking into account the three received segments (indicated by the three duplicate ACKs). As soon as an ACK arrives acknowledging reception of new data, congestion window is reset to the value of "ssthresh" and the TCP sender re-enters the congestion avoidance (AIMD) phase.

What has been described so far about TCP is referred to as "TCP dynamics". Any other transport protocol that implements these dynamics is referred to as "TCP-compatible". "TCP-friendly", on the other hand, refers to a transport protocol that only converges on the same flow rate as TCP's, without implementing the TCP dynamics. It is called "friendly", since it is assumed that it behaves "fairly" in competition with TCP flows at the network bottlenecks. As the number of applications using alternative transport protocols have increased significantly, some efforts have been made in specifying "TCP-friendliness" [8, 9]. In specifying "TCP-friendliness", developed models which estimate the throughput that can be achieved by a TCP flow have been used; with the simplest model estimating throughput based on MSS (Maximum Segment Size), RTT (Roundtrip time) and p (constant probability of random packet loss) which, using the constant $C = \sqrt{\frac{3}{2}}$, will result in TCP throughput estimated by $\frac{MSS}{RTT}\frac{C}{\sqrt{p}}$ [10].

3 Explicit Congestion Notification (ECN)

Notification of congestion to end-systems has been traditionally done by routers dropping packets. "Explicit Congestion Notification (ECN)"[11] has been proposed to add a separate mechanism for routers to explicitly notify end-hosts of congestion without dropping packets and it works as follows. ECN uses two bits in the IP packet header; the last two bits of "Type of Service" (TOS) byte with the first six bits used for "Differentiated Services CodePoint (DSCP)"[4]. The two bits in the IP header lead to four possible ECN codepoints, use of which is described as follows. Codepoint "00" indicates "Not-ECN-Capable Transport (Not-ECT)" meaning the ECN is ignored and is used neither by the end nodes nor by the routers in the path; this is the default value for the two bits. Codepoints "01" and "10" both indicate "ECN-Capable Transport (ECT)" and are referred to as "ECT(1)" and "ECT(0)", respectively. Finally, codepoint "11" indicates "Congestion Experienced (CE)"; therefore, "ECN-CE" marking replaces packet dropping in the router, if the router wants to

[4] IP version 6 uses a similar arrangement to IPv4 described in the text.

indicate congestion. Apart from IP header, two bits in the TCP header are also reserved for ECN; one flag for "Congestion Window Reduced (CWR)" and one for "ECN-Echo (ECE)" (explanation of their use follows).

When an end-node is ECN-capable, it sets the ECN codepoint in the IP header of the packet to either ECT(1) or ECT(0). A congested router which wants to send congestion notification sets the ECN codepoint to CE, instead of dropping the packet which is the default treatment of Non-ECT packets. Upon receipt of this CE marked packet by the TCP receiver, it sets the ECN-Echo flag in the TCP header in the next packet that it sends back to the sender (e.g. in an ACK). Therefore, as soon as the sender receives the ECN-Echo flagged packet, it will know about the congestion and will react as if a packet has been dropped (the original congestion signal) by halving its send window. The sender will also set the Congestion Window Reduced (CWR) flag in the TCP header of the next outgoing packet to indicate that it has received and reacted to the congestion signal.

An aspect which we deliberately delayed to explain in the above paragraphs, is the fact that we mentioned that ECN uses two codepoints, "ECT(1)" and "ECT(0)", to signal one state: "ECN-Capable Transport (ECT)". This effectively means that in the basic ECN, only 3 codepoints carry useful information; with one being unused. However, "(ECN)-nonce" [12] changes that, by putting in use both "ECT(1)" and "ECT(0)" codepoints by adding another functionality: protecting against accidental or deliberate concealment of ECN-CE marked (or dropped) packets from the TCP sender. With "(ECN)-nonce", TCP receivers have to report correctly whether or not the received packets have been ECN-CE marked in the path. Without this mechanism, i.e. with basic ECN, receivers could lie about the real congestion in the path (by downplaying it) and fooling senders into continuing to send with the same rate, thereby gaining an unfair advantage over other correctly-behaving receivers. Apart from the two ECN codepoints, "(ECN)-nonce" reserves and uses a flag in the TCP header. In what follows, we provide an overview of "(ECN)-nonce".

The sender encodes a sequence of random one-bit nonces onto the codepoints ECT(1) and ECT(0), i.e. ECT(0) represents a nonce bit of 0 and ECT(1) represents a nonce bit of 1. This way, the sender can indicate ECN capability, all the while using this opportunity to encode its random bits. We remind from ECN description above that a router who wants to signal congestion, overwrites the two ECN bits with the ECN-CE codepoint. This means that at the receiver, the encoded random bit by the sender is lost, if the packet is ECN-CE marked. The receiver calculates a sum of the values of the received nonce bits and returns the sum value to the sender using a TCP header flag, the nonce sum (NS) bit. If packets arrive without being ECN-CE marked (or dropped),

the sum value will match the value held at the sender, otherwise, the two sum values will differ and the receiver has a very low chance of being able to repeatedly guess the correct sum values. Consequently, the sender is able to verify whether or not the receiver reports correctly the reception of ECN-CE marked (or dropped) packets[5].

4 Active Queue Management - AQM

After having set the scene about congestion control and the issues at play, we introduce the prominent Active Queue Management (AQM) schemes. Congestion avoidance schemes maintain the network in a region of low delay and high throughput. Congestion avoidance can be approached by focusing on end-to-end schemes, router-based schemes (i.e. AQM) or, ideally, combination of both.

4.1 Random Early Detection - RED

Random Early Detection (RED) [13] implements the congestion avoidance mechanism by controlling the average queue size. Compared to Drop-Tail, RED avoids global synchronization of TCP connections decreasing their windows at the same time and, contrary to Drop-Tail, RED does not have bias against bursty traffic. RED uses randomization to calculate the drop probability for each arriving packet and drops the packet with this probability which is about proportional to the corresponding connection's bandwidth through the gateway.

RED uses two separate algorithms to accomplish its stated objectives: one algorithm to calculate the average queue size and another algorithm to calculate packet drop (or marking) probability. The principal parameter in the algorithm to calculate average queue size is w_q, the queue weight, which basically determines the size and duration of bursts that can be accepted at the gateway. w_q determines how closely "average queue size (avg)" follows the "current queue size"; a small w_q

[5] While "(ECN)-nonce" certainly adds an interesting functionality to the ECN scheme, it has been mentioned in this section for another reason. "Re-ECN", an scheme presented in Section 5, redefines how ECN codepoints are interpreted and used. The designers of "re-ECN" has not managed to accommodate "(ECN)-nonce" in the new interpretation of the codepoints. At the same time, "(ECN)-nonce" is currently held at "Experimental" status, since its final standardization renders implementation of "re-ECN" impossible. This has led to a fierce discussion within the IETF between the proponents of each scheme, which we felt we have to mention, considering that we refer to "re-ECN" on many occasions later on in the document.

means that a short-term increase in the current queue size, due to a burst, will have little effect in increasing the average queue size(*avg*).

After having calculated *avg* upon packet arrival, the second algorithm uses the value of *avg* to calculate the drop probability and decide whether or not to drop the packet. In calculating the drop probability, few parameters are at play: max_{th}, min_{th} and max_p (queue size maximum and minimum thresholds and maximum drop probability, respectively). If the calculated *avg* is below the min_{th}, then the drop probability is zero. If *avg* is between min_{th} and max_{th}, then a drop probability is calculated which is between 0 and max_p. If, however, *avg* is higher than max_{th}, then the drop probability is 1. A later modification to RED to provide a less aggressive drop probability function is the following: for *avg* value between max_{th} and $2 * max_{th}$ (the "gentle region"), the drop probability is linearly increased from max_p to 1.

Final notes on RED include the ability of RED to measure queue size in bytes instead of packets. Measuring in bytes entails modification of drop probability function and taking into account the arriving packet size in relation to the maximum allowable packet size. This in turn implies that larger packets have higher chances of being dropped than smaller packets (i.e. packets like ACKs are less likely to be dropped). Part of the RED gateway configuration is setting values for max_{th} and min_{th} which depends on the desired "average queue size(*avg*)". Deciding on an optimal value for this latter parameter, *avg*, to simultaneously maximize throughput and minimize delay is not a trivial issue and depends on the traffic characteristics as well as on network's physical characteristics.

4.2 RED with Preferential Dropping (RED-PD)

Short Round-Trip Time (RTT) TCP flows (i.e. high throughput TCP flows) and flows that fail to use end-to-end congestion control are two categories of flows that unfairly take too much of the available bandwidth at FIFO routers. To remedy this problem, "RED with Preferential Dropping (RED-PD)" [14] has been proposed. In "max-min fairness", flows are ordered according to their demands, then flows with minimal demands are satisfied before sharing the remaining resource (bandwidth at the router's output interface) between high-demand flows. Implementing a full "max-min fairness" scheme requires per-flow scheduling and keeping state for all flows traversing the router; hence considered complex and near-impractical. RED-PD proposes a "partial flow state" approach in which state is kept only for high-bandwidth flows. Therefore, RED-PD achieves "max-min fairness" along with keeping the simplicity of FIFO queueing.

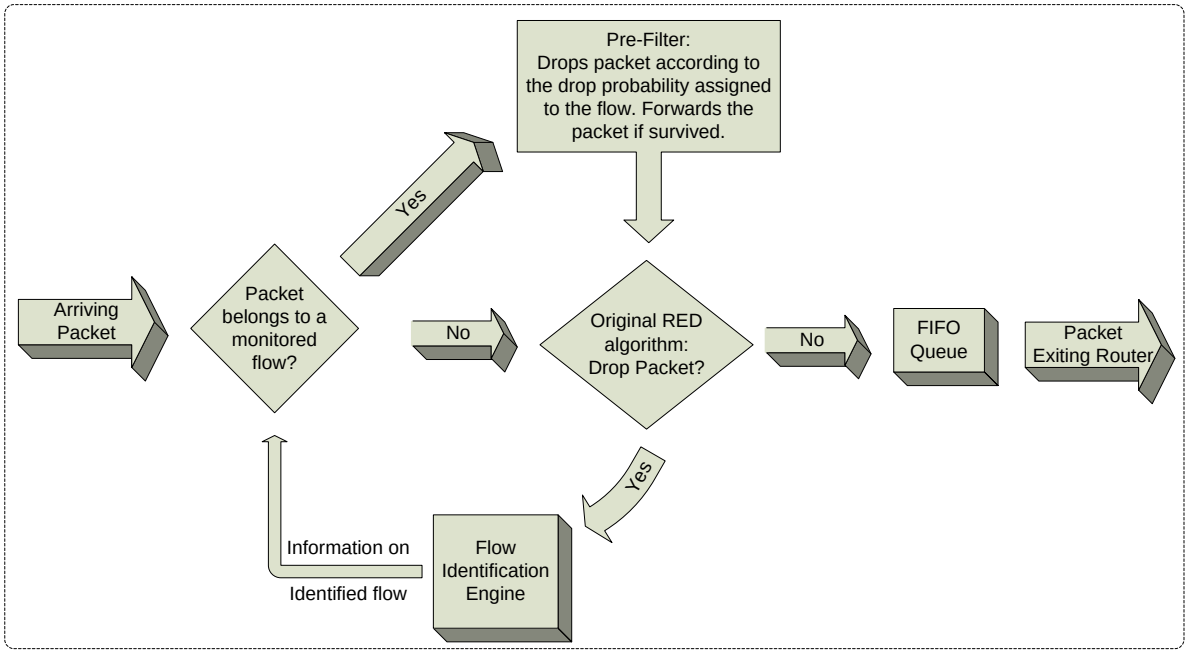

Figure 1: Architecture of a RED-PD router

In RED-PD scheme, high-bandwidth flows are identified using RED's packet drop history, which is considered to be a reasonable sample of the incoming traffic [14]. RED-PD identifies flows with bandwidths above a "target bandwidth", monitors them and controls their bandwidth at a pre-filter (see Figure 1) placed before the output queue by probabilistically dropping packets. The drop probability, in the pre-filter, is calculated by the identification method and it is made sure that the rate of each flow going to the output queue does not exceed the "target bandwidth".

For identification of high-bandwidth flows, as mentioned before, a "target bandwidth" is used. This "target bandwidth" is chosen as the rate (pkts/sec) of a reference TCP flow with a "target RTT (R)" and the current drop probability p of the RED algorithm (see Figure 1, block in the middle). This reference TCP flow has approximately an average rate of $\frac{\sqrt{1.5}}{R*\sqrt{p}}$ (pkts/sec) [14, 15]. In the deterministic model [15], a "TCP Congestion Epoch" has one packet drop and therefore has $\frac{1}{p}$ packets in total (counting the dropped packet and the successfully forwarded packets). Therefore, based on the mentioned formula for TCP flow average rate, the rate of packet drops is $p * \frac{\sqrt{1.5}}{R*\sqrt{p}}$ or $\frac{\sqrt{1.5*p}}{R}$. We can infer automatically the "Congestion Epoch Length (CL)"(in sec.) to be $\frac{R}{\sqrt{1.5*p}}$, which indicate the period in which at least one packet is dropped on average, if a flow sends at a rate higher than

$\frac{\sqrt{1.5}}{R*\sqrt{p}}$. RED-PD algorithm maintains packet drop history for $K*CongestionEpochLength(CL)$ seconds (with K being a small integer) and identifies a flow as "high-bandwidth" if it has K or more drops during this period.

As has been clear in the previous paragraph, RED-PD designers have transformed setting a value for "target bandwidth" to setting a value for "target RTT (R)", as a RED-PD configuration decision. In what follows, we discuss the significance of "target RTT (R)" as a configuration parameter. Setting the "target RTT (R)" to a high value leads to more flows being monitored; hence more state kept and nearing full "max-min fairness". At the other side of the spectrum, choosing a small value for "target RTT (R)" means RED-PD is having little impact at the router in the context of flow rate regulation. If the goal is the control of only very short RTT TCP flows along with overly-misbehaving users (very high-bandwidth flows with no end-to-end congestion control), then "target RTT (R)" should be chosen as a small value accordingly.

"Preferential Dropping" is done in the pre-filter (see Figure 1, upper block), in which packets from the monitored flows are dropped with a probability dependent on how much the flow rate has exceeded the "target bandwidth". As is clear from Figure 1, the unmonitored flows (the flows with low or acceptable level of bandwidth) bypass the pre-filter and go directly to the output queue and will be subjected to the normal RED algorithm. If the RED's average queue size is low however, meaning that there is low overall traffic demand, RED-PD stops dropping packets in the pre-filter, therefore the whole RED-PD algorithm degrades to a simple RED mechanism.

In closing our notes regarding RED-PD, we emphasize the fact that, as the identification engine does not run in the forwarding fast path, storage and processing of drop history have insignificant overhead. This is due to simple processing and the maintained state which is a sparsely-populated classifying data structure that is used to lookup the dropping probability (to be used in the pre-filter) for any monitored flow.

4.3 Discussions and Comparisons

The main weakness of RED AQM scheme is that the average queue size is dependent on the congestion level and initial parameters setting. Specifically, the average queue size is around min_{th}, when max_p has been set to a large value and/or the level of congestion is low. On the other hand, the average queue size is around max_{th}, when max_p has been set to a small value and/or the level of congestion is high. This leads to a situation in which network operators are unable to estimate

the queueing delays in advance. A closely related problem to this issue is the similar dependence of throughput. Namely, RED leads to low throughput and high drop rate when average queue size is around and above max_{th}, which is unavoidable at times due to unpredictable congestion levels. Adaptive RED [16] (based on some related earlier works) makes minor changes to the basic RED algorithm to remove its sensitivity to initial parameters setting; hence achieving a target average queue length in a variety of traffic scenarios.

As discussed above, average queue size and overall performance of RED depend on how correctly parameters max_p and w_q have been chosen considering the expected congestion level. Adaptive RED basically auto-tunes max_p dynamically based on current congestion level (by measuring queue length) and leads to average queue size maintaining a value within a target range between min_{th} and max_{th}. Adaptive RED also sets a value for w_q automatically based on the link speed. The authors have conducted simulations to demonstrate the effectiveness of this design in which automatic adaptation of max_p indeed leads to a steady average queue size by tuning the relationship between average queue size and packet dropping probability. Given that Adaptive RED sets some parameters automatically, the only configuration left is the choice of target average queue size which, as has been mentioned before, is a question of policy regarding the trade-off between throughput and delay.

Apart from RED (and other RED-based algorithms), there are two other major AQM designs: Proportional-Integral (PI) Controller [17] and Random Exponential Marking (REM) [18]. While these designs have also shown solid performances compared to RED, they have nevertheless had less traction and industry adoption (e.g. implementation in Cisco routers). Furthermore, these two designs have not received yet the intense scrutiny in variety of simulation scenarios and actual implementations as has been the case for RED. In what follows, we give a very brief overview of PI controller and REM. Note that both PI and REM solely deal with TCP flows; their performance in dealing with UDP flows (i.e. where flows are not responsive to congestion notification) is not known.

The PI designers [17] have used classical control system techniques to develop AQM mechanisms supporting TCP flows. The idea is inspired by, and tries to address, two reported limitations of RED: 1) Tradeoff between stability and speed of response 2) Coupling of queue length and loss probability which results in a flow, facing an overloaded system, to experience high delay as well as high loss.

Proposing to decouple marking probability from queue size, an integral controller is used which

attempts to keep the queue size around a reference value q_{ref} independent of the load level. It has been suggested that the Proportional-Integral (PI) controller: 1) is relatively insensitive to load level variations and attempts to keep the queue size to a set value q_{ref} 2) provides faster response times and less oscillations 3) allows delay to be tuned with a single parameter, q_{ref}, unlike RED, in which delay is a function of current load and several parameters. 4) is capable of operating at both high utilization and low delay, unlike RED, in which one can be achieved at the expense of losing the other. In summary, the proportional term of the PI controller is responsible for the fast response time and providing stability and its integral term is responsible for the steady-state regulation of queue size around the reference value q_{ref}, even if there are load variations (i.e. variations in number of TCP sessions and their RTT values).

Random Exponential Marking (REM) [18] is another AQM scheme which is built on the idea that congestion measure (which tracks the traffic demand, i.e. number of TCP flows, etc.) should be decoupled from performance measures (delay and loss). It is therefore capable of offering both high utilization and low delay at the same time. Although the design process has been different, REM and PI controller are equivalent schemes.

REM has two key features: "Match rate clear buffer" and "Sum prices". "Match rate clear buffer" refers to the fact that REM sends feedbacks to sources, and sources in turn adapt their sending rate, to control the aggregate arrival rate to the router, which results in keeping the aggregate arrival rate around the link capacity. This control happens while keeping the queue length around a target value and is independent of the number of TCP flows. "Sum price" ("price" is REM's congestion measure) serves to provide user with a way to estimate the overall congestion in case there are multiple congested links in the path.

The dropping probability is an exponentially-increasing function of the calculated "price" at each output queue. This leads to the fact that as the packet moves along a path, an end-to-end dropping probability can also be defined which will be an exponentially-increasing function of "sum" of all the "prices" along the path which is the direct result of defining this "exponential" function. When a source sees a particular drop rate, it can then infer (using the assumption of end-to-end exponential drop probability) the end-to-end "Sum prices" or overall congestion along the path.

As mentioned, each output queue keeps a congestion measure, called "price", which is used to determine marking/dropping probability and gets updated based on the difference between arrival rate and link capacity and the difference between current queue size and the set queue size target. According to this notion of "price", as the arrival load increases, the value of price increases and a

stronger congestion signal is sent to the senders while the mean queue size is stabilized around the set target.

The three major AQM schemes (Adaptive RED, PI and REM) have been the subjects of an empirical study [19] for comparing their performances (with and without ECN support) with end-to-end response time distribution of HTTP request/response exchanges being the primary performance metric (web users perceived performance). In summarizing the results of the empirical study, we start by pointing out that Adaptive RED shows its best performance in byte-mode and without ECN. With this configuration, Adaptive RED outperforms REM, PI and Drop-Tail at 90% load (all with or without ECN, operating in their recommended byte-mode). However, at 98% load, REM and PI with ECN support outperform ARED (byte-mode without ECN). This leads to the conclusion that Adaptive RED (byte-mode) could be the right choice for AQM if the operator wishes to avoid ECN implementation and have high link capacity utilization (near-saturation level).

5 Fairness between Economic Entities

Although we advocate the view of providing fairness between users instead of flows, up to this point in this report, the dominant view of providing fairness between flows as a goal has been present in various contexts, faithfully representing the views of the designers of those schemes. This has been intentional, as we have concentrated our arguments in favor of providing fairness "between users" rather than "between flows" in this section.

Striving to achieve "flow rate fairness" has been present in almost all queueing and congestion control mechanisms. This presence has taken the form of either requiring that all applications use TCP or otherwise be TCP-friendly on the client side[6], or implementing queueing schemes such as Fair Queueing [20], Weighted Fair Queueing [21] and Core-Stateless Fair Queueing [22], or the previously-discussed AQM schemes, on the router side. However, this "obvious and natural" goal can hardly be any more irrelevant in today's Internet. The idea of trying to provide "fairness" is meaningful when considered in the context of relationships between "real people", "businesses", etc. (hereafter referred to as "economic entities" or "users"). At the early days of Internet, trying to provide "flow rate fairness" largely meant achieving fairness between the economic entities as well. The nature of applications run in the Internet, in terms of number and duration of flows, was

[6]Note that we still consider congestion avoidance aspect of TCP and TCP-friendly protocols a desirable feature for the simple reason that it tries to avoid congestion. However, it is far from "fair" to users in practice.

far more harmonious than what is seen in today's Internet. This meant that most queueing and congestion control mechanisms took care of the issue of providing fairness between "economic entities", just by trying to provide fairness between "flow rates". This fundamental misconception has gone largely unnoticed, until heavily exploited by the P2P applications. P2P applications are known to: 1) create numerous flows and 2) use them in parallel for long periods of time. As a result, P2P applications have fully exploited the two loopholes present in the idea that "flow rate fairness" naturally leads to "fairness between users".

Under current queueing and congestion avoidance/control schemes, which try to achieve "user fairness" through "flow rate fairness", P2P applications gain a "grossly-unfair" advantage over other more traditional Internet applications like HTTP. This is due to the fact that those schemes try to provide "fairness" by allocating equal bandwidth to competing flows in every bottleneck in the network. This type of allocation is myopic: it makes decision based on the instantaneous status of the queue and does not take into account the (sufficiently long) history (i.e. how long the competing flows have been active in the past). Furthermore, this type of allocation does not take into account the number of flows that each application has created[7].

Recently, Briscoe [23–27], a researcher in BT/UCL, has extensively discussed the issues mentioned above and has designed a scheme called "re-ECN"[28] to achieve what he refers to as "cost fairness". "Cost fairness" is built on the notion of putting price on the congestion that users cause in the network and its implementation entails minor changes to TCP/IP along with addition of other policing/dropping devices to the network. We now concentrate on "re-ECN", Briscoe's solution to the Internet's congestion problem.

5.1 Re-ECN

The "re-ECN" scheme [28] is a concrete proposal as to how price can be put on the congestion that users cause, achieving "cost fairness" as a result[8]. As explained in Section 3, "ECN" will enable to see the path congestion in the receiver side. The receiver, in turn, informs the sender, through a TCP flag, about the experienced congestion on the path. Finally, the sender "can" take action to

[7] Some of the AQM mechanisms try to keep a certain amount of history, mitigating the myopic aspect. However, they neglect to, or cannot, keep track of number of flows per application.

[8] The designers of "re-ECN" believe that this scheme effectively closes the loopholes exploited by the P2P traffic, so "fair" use of the network by all traffic types including the P2P traffic follows naturally after deployment of "re-ECN".

correct the situation by reducing its rate. The obvious problem is that the network is at the mercy of the sender and cannot force the sender to reduce its rate; only an egress dropper, just before reaching the receiver, can stop the rogue flow by monitoring the ECN-CE marks and consequent sender rate response to the congestion indication, but the congestion damage is already done in the network[9]. "Re-ECN" corrects this situation by exposing the congestion information at the sender's side, so an ingress policer can take action before any damage is done in the network. Furthermore, "re-ECN" is designed to be able to control single datagram flows as well, as it can operate with just sender support; the scheme works best though if both sender and receiver have re-ECN-capable transports. Hereafter, we explain the mechanics of "re-ECN".

"Re-ECN" requires the last undefined bit (RE flag) in the IPv4 header. RE flag, in combination with the 2-bit ECN field, create eight codepoints[10]. Routers along the path do not need to be re-ECN-capable; ECN-capability is assumed to be present though. The idea is that as the flow's packets traverse the network, they are ECN-CE-marked in ECN-capable routers, if the routers experience congestion. The receiver then "counts" the number of received marks and reports back this number (CE-no.) to the sender using TCP. The sender then marks "CE-no." number of its outgoing packets by setting the RE flag of their IP headers, effectively indicating to the network (right at the ingress point) the amount of the congestion expected downstream.

When a sender first starts to send packets, it does not know the congestion that will be experienced. Therefore, it uses the "feedback not established" codepoint. The network however, treats this initial phase as if the sender is declaring downstream congestion, to protect against misbehaving senders. Once the feedback is established from the receiver, the sender sets the RE flag to 1 in the next packets that it sends. If the receiver reports back to the sender that some number of packets have been received CE-marked, the sender sets the RE flag to 0 in an equivalent number of packets that is going to send. Hereafter, we use the term "blanking" to refer to process of setting RE flag to 0. Likewise, "RE blanking fraction" refers to the fraction of octets in a stream of packets in which the RE flag have been "blanked", or set to 0.

Figure 2 (adapted from [28]) depicts how re-ECN allows routers along the path to be able to measure not only the upstream congestion (which can be achieved by only ECN), but also estimated

[9] Another scenario would be that the receiver does not report at all the received congestion marks to the sender, hoping to maximize its reception rate.

[10] One of re-ECN's codepoints is an alternative use of the codepoint ECT(1) used to implement ECN-nonce; see the footnote in Section 3

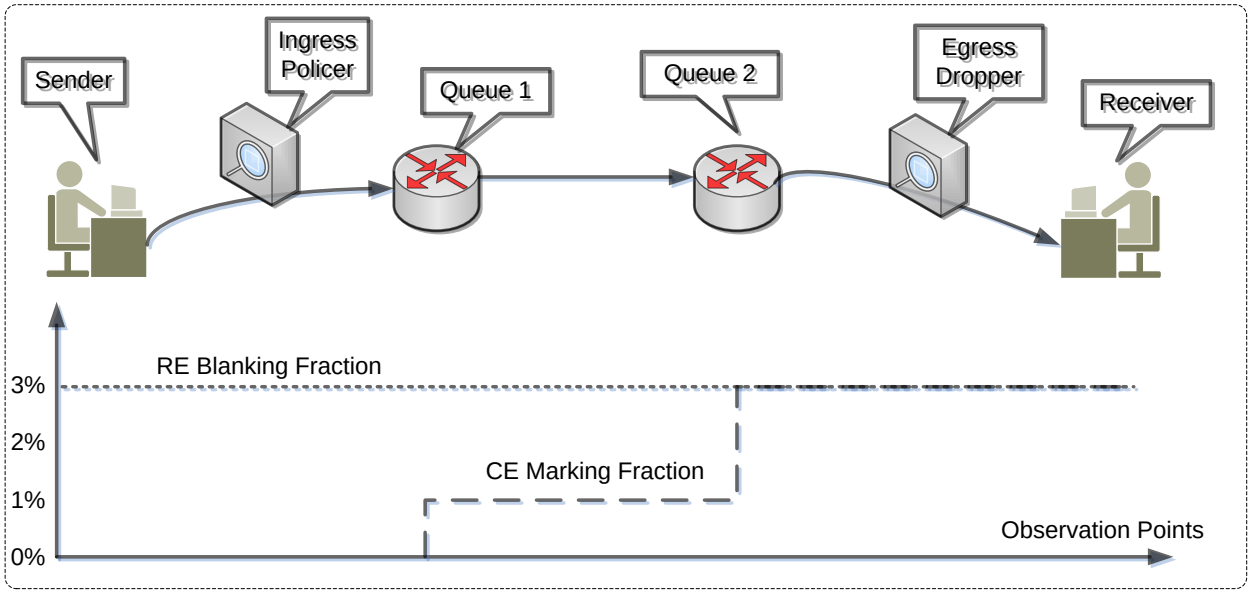

Figure 2: Re-ECN: An approximative 2-queue example

downstream congestion (which is made possible by re-ECN). In the case of Figure 2, receiver notices that about 3% of packets have been congestion (CE) marked and reports this to the sender. The sender then "blanks" 3% of its outgoing packets to match the 3% CE fraction seen at the receiver. Note that the RE flag is not changed in the network; it is only set by the sender. This effectively means that at any observation point in the path, it is possible to measure the upstream congestion (by measuring the fraction of packets that have been CE marked), the end-to-end path congestion (by measuring the fraction of packets that have been "blanked" by the sender) and the downstream congestion (by subtracting upstream congestion from whole path congestion).

We introduce few other terms: "positive packet" or "negative packet", meaning a "blanked packet" or "CE marked packet", respectively. Equivalently, "positive flow" means a flow that carry more positive packets than negative ones on average, or put another way, it has a positive downstream congestion metric (in the same manner, a "negative flow" is defined.). Note that the "worth" of each "positive packet", for example, depends on the "size" of the packet, meaning that a large packet contributes more to the "positiveness" of the flow. As mentioned, amount of congestion can now be measured in every point in the path with re-ECN. Therefore, two key elements, "ingress policer" and "egress dropper" are deployed to make sure that flows do not go consistently

negative[11]. These two pressures ensure that the receiver honestly reports to the sender the true level of congestion in the path and that the sender reacts responsibly to adapt to this level of congestion. Note that "ingress policer", like any other observation point in the path, is aware of level of congestion that the sender is causing in the whole path; whether or not the policer rate-controls the sender depends on the type of contract between the sender and the access network operator.

[11] A flow that is consistently negative, understates the congestion experienced in the network. This behavior is sanctioned by the egress dropper. If a flow is consistently positive, it overstates the congestion experienced, which is only against end-nodes' interests; leading to the sender using its "congestion volume quota" quicker.

6 UARA

[This section is added to this Tech Report in 2011 to mention a paper due to its relevancy]

Masood Khosroshahy (2011), *"UARA in edge routers: An effective approach to user fairness and traffic shaping"*, International Journal of Communication Systems (Wiley), doi: 10.1002/dac.1262

Abstract:

The ever-increasing share of the peer-to-peer (P2P) traffic flowing in the Internet has unleashed new challenges to the quality of service provisioning. Striving to accommodate the rise of P2P traffic or to curb its growth has led to many schemes being proposed: P2P caches, P2P filters, ALTO mechanisms and re-ECN. In this paper, we propose a scheme named "UARA:User/Application-aware RED-based AQM" which has a better perspective on the problem: UARA is proposed to be implemented at the edge routers providing real-time near-end-user traffic shaping and congestion avoidance. UARA closes the loopholes exploited by the P2P traffic by bringing under control the P2P users who open and use numerous simultaneous connections. In congestion times, UARA monitors the flows of each user and caps the bandwidth used by "power users" which leads to the fair usage of network resources. While doing so, UARA also prioritizes the real-time traffic of each user, further enhancing the average user quality of experience (QoE). UARA hence centralizes three important functionalities at the edge routers: (1) congestion avoidance; (2) providing user fairness; (3) prioritizing real-time traffic. The simulation results indicate that average user QoE is significantly improved in congestion times with UARA at the edge routers.

Uara resources available from: http://www.masoodkh.com/uara/

References

[1] S. Floyd. Congestion control principles. IETF RFC 2914, September 2000.

[2] John Nagle. Congestion control in ip/tcp internetworks. IETF RFC 896, January 1984.

[3] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329. ACM, 1988.

[4] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. IETF RFC 2581, April 1999. (Partially updated by RFC 3390).

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. IETF RFC 2616, June 1999.

[6] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the internet. IETF RFC 2309, April 1998.

[7] M. Duke, R. Braden, W. Eddy, and E. Blanton. A roadmap for transmission control protocol (tcp) specification documents. IETF RFC 4614, September 2006.

[8] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Tcp friendly rate control (tfrc): Protocol specification. IETF RFC 5348, September 2008.

[9] J. Widmer, R. Denda, and M. Mauve. A survey on tcp-friendly congestion control. *IEEE Network*, 15(3):28–37, 2001.

[10] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, 1997.

[11] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ecn) to ip. IETF RFC 3168, September 2001.

[12] N. Spring, D. Wetherall, and D. Ely. Robust explicit congestion notification (ecn) signaling with nonces. IETF RFC 3540 (Experimental), June 2003.

[13] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[14] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proc. Ninth International Conference on Network Protocols*, pages 192–201, 11–14 Nov. 2001.

[15] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[16] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive red: An algorithm for increasing the robustness of red's active queue management. Technical report, AT&T Center for Internet Research at ICSI, 2001.

[17] C.V. Hollot, V. Misra, D. Towsley, and Wei-Bo Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proc. IEEE Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2001*, volume 3, pages 1726–1734, 22–26 April 2001.

[18] S. Athuraliya, S.H. Low, V.H. Li, and Qinghe Yin. Rem: active queue management. *IEEE Network*, 15(3):48–53, May–June 2001.

[19] Long Le, J. Aikat, K. Jeffay, and F.D. Smith. The effects of active queue management and explicit congestion notification on web performance. *IEEE/ACM Transactions on Networking*, 15(6):1217–1230, Dec. 2007.

[20] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, pages 1–12. ACM, 1989.

[21] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[22] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Trans. Netw.*, 11(1):33–46, 2003.

[23] Bob Briscoe. A fairer, faster internet protocol. *IEEE Spectrum*, December 2008.

[24] Bob Briscoe. Flow rate fairness: dismantling a religion. *SIGCOMM Comput. Commun. Rev.*, 37(2), 2007.

[25] Arnaud Jacquet, Bob Briscoe, and Toby Moncaster. Policing freedom to use the internet resource pool. In *Proc Workshop on Re-Architecting the Internet (ReArch'08)*. ACM, December 2008.

[26] Bob Briscoe, Arnaud Jacquet, Carla Di Cairano-Gilfedder, Alessandro Salvatori, Andrea Soppera, and Martin Koyabe. Policing congestion response in an internetwork using re-feedback. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2005.

[27] B. Briscoe, T. Moncaster, and L. Burness. Problem statement: Transport protocols don't have to do fairness. IETF Internet-Draft [draft-briscoe-tsvwg-relax-fairness-01], July 2008. (Work in Progress).

[28] B. Briscoe, A. Jacquet, T. Moncaster, and A. Smith. Re-ecn: Adding accountability for causing congestion to tcp/ip. IETF Internet-Draft [draft-briscoe-tsvwg-re-ecn-tcp-06], July 2008. (Work in Progress).